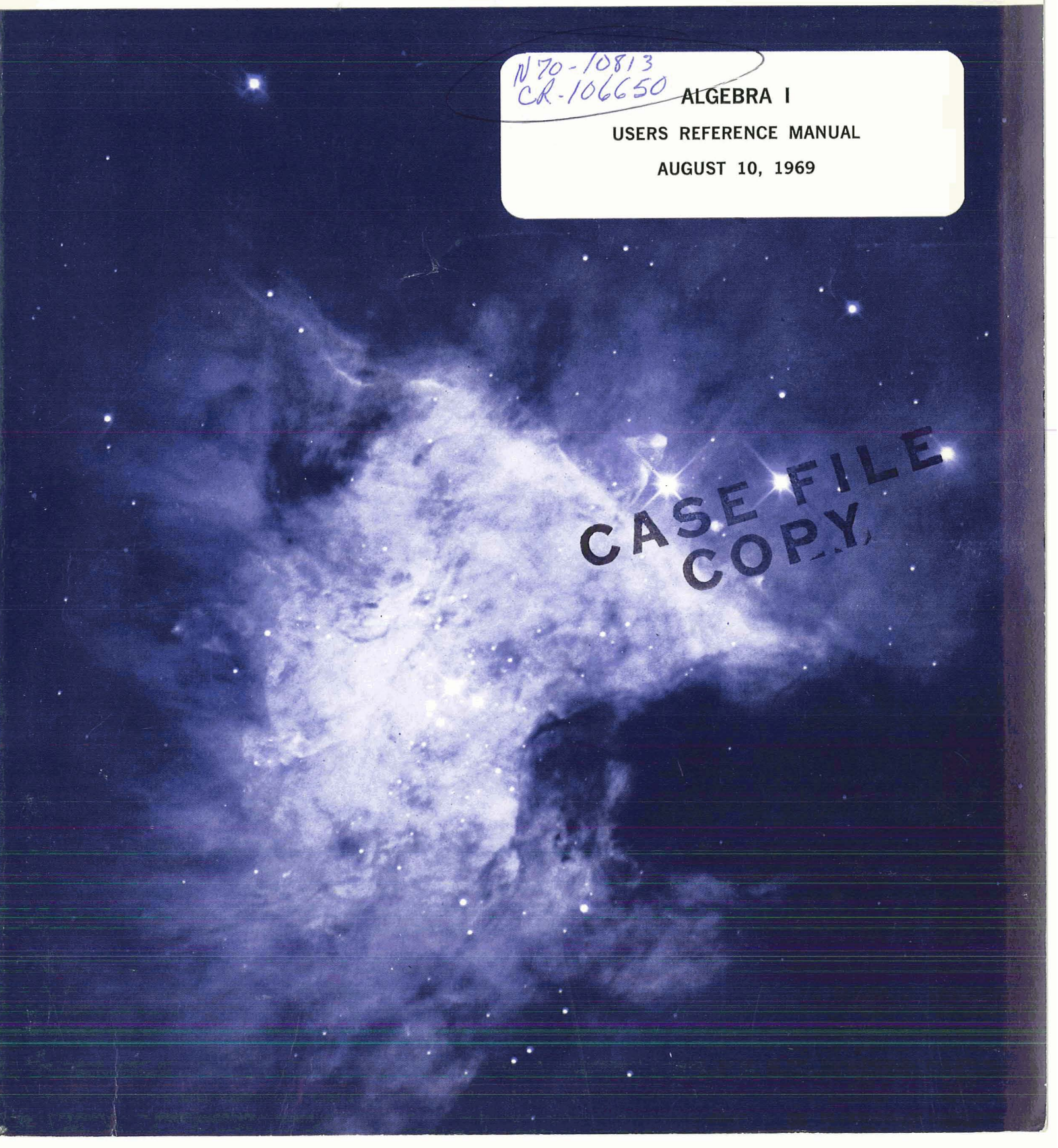
UNIVERSITY OF MINNESOTA

SPACE SCIENCE CENTER

ALGEBRA I

USERS REFERENCE MANUAL

AUGUST 10, 1969



ON THE COVER — The area of the heavens around the Orion Constellation, shown in the cover photograph made through the 120-inch telescope of the Lick observatory, is also the region of observations with an infrared telescope developed by University of Minnesota astro-physicists. The infrared sensory equipment reveals stellar bodies that could not be studied by conventional telescopes, and it is expected to provide data on the birth of stars.

ALGEBRA I

USERS REFERENCE MANUAL

Edition 1

F. N. Bailey

J. Brann

R. Y. Kain

Department of Electrical Engineering

University of Minnesota

Minneapolis, Minnesota 55455

August 10, 1969

The work reported herein was supported by the National Aeronautics and Space Administration/University Sustaining Grant NGL24-005-063

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

1. INTRODUCTION

The ALGEBRA I language is an interactive language designed for the symbolic manipulation of polynomials and rational functions. It is written in SNOBOL 67, a dialect of SNOBOL3, and runs on the SDS 940 under the COMSHARE* W-series Executive System. Due to space limitations ALGEBRA I will not run in the current COMSHARE version of SNOBOL. However, it could be readily modified to run under any SNOBOL3 or SNOBOL4 system with sufficient space available.

1.1 The ALGEBRA Manipulation Problem

Algebra manipulation is a relatively new area of computer applications where one attempts to take advantage of the symbol manipulating capabilities of the digital computer rather than its arithmetic capabilities. Since there are significant technical problems where the algebraic manipulation required in analysis is quite difficult, it is natural to seek the aid of the digital computer in carrying out the analysis. However it should be emphasized that the goal of this computation is usually manipulation rather than solution. The computer performs symbolic manipulations as instructed by the programmer, but it does not make decisions about what manipulations are needed. This distinction between problem solving systems and manipulation systems is important. The former may require the latter but the converse is not true.

---

*COMSHARE INC., Ann Arbor, Michigan, sells time-shared service on the SDS 940.

Since the goal of algebra manipulation is to aid the human in complex symbol manipulation tasks, close cooperation between man and machine is desired. Most of the decisions are to be made by the man on the basis of data provided by the machine. Thus an interactive or on-line situation is most productive. At present this sort of environment is available at reasonable expense only through the use of a time-shared system. In the design of the ALGEBRA language the importance of man-machine interaction has been a prime consideration and the operation of the language on a time-shared facility has been considered a necessity.

The basic idea of computer-aided algebra manipulation has arisen in a number of different places. Early languages were the FORMAC [2] language, a superset of FORTRAN, developed at IBM, and the ALPAK [3] and ALTRAN [4] languages developed at Bell Telephone Laboratories. In recent years a number of other languages have been reported but almost all of these are batch processed languages lacking the man-machine interaction so essential to the success of this approach to problem solving. Notable exceptions are the AMTRAN language [5] which runs on a dedicated computer system but does provide for user interaction and the MATHLAB system [6] on the Project MAC computer at MIT. ALGEBRA I is apparently the first general purpose algebra manipulation language running on a commercial time-shared computer system.

1.2 Applications

Algebra manipulation languages are useful in solving problems which involve long complex algebraic manipulation steps too tedious for practical human hand computation. Since the machine can perform error free manipulations of great complexity at high speeds, it offers considerable extension of human performance. Applications of this capability have been reported in:

1) determining equilibrium conditions in structural mechanics
2) perturbation solutions of nonlinear ordinary differential equations
3) orbit calculations
4) studies of high energy particle interactions.

In most of the previous applications the required manipulation steps were known but were too complex for hand calculations. Thus an algorithm for the required algebraic manipulation could be programmed on a batch-processing computer system to obtain the desired results.

There exist other classes of problems where the required manipulative steps are not known a priori and an algorithm for a batch processed language cannot be described in advance. For treating such problems an interactive system is required so that the operator can view intermediate results and make decisions about further processing steps on the basis of these intermediate

results. For these problems the algorithm must be developed hueristically as the solution proceeds--possibly by trial and error--under control of the man-machine team. It is for this second class of algebra manipulation problems that the ALGEBRA system is specifically structured. User-problem and man-machine interaction have been given major consideration in all phases of the system design.

1.3 Outline of Language Features

The ALGEBRA I language is designed for the manipulation of polynomials and rational functions having numbers and parameters in the coefficients and exponents. Polynomials and rational functions are assigned names on input and may be referenced by name in all further manipulation. Operations available to the user are: addition, subtraction, multiplication, division and differentiation of rational functions, plus integration of polynomials. The user may also make substitutions of rational functions for the variables in other rational functions.

The input format is very flexible with normal mathematical notation used wherever possible. Input statements may be executed line by line or saved as blocks or programs to be executed at a later time. Line editing features for stored blocks of statements are also provided. Specially designated blocks called procedures

provide user defined "subroutines" of algebra manipulation statements which can be called with rational functions as arguments.

No previous knowledge of SNOBOL or any other computer language is required. It is intended that anyone familiar with college algebra or calculus can learn to use the ALGEBRA system in a negligible period of time. However, a more sophisticated user with some knowledge of SNOBOL can include SNOBOL statements in his ALGEBRA programs to provide for complex tests and conditional branching.

1.4 Organization of the Manual

The ALGEBRA I manual is intended to provide potential users with a detailed description of the language and the required data format. Section 2 covers data format, Section 3 covers the syntax and semantics of the ALGEBRA language, Section 4 describes the operation of the ALGEBRA I system and Section 5 gives some example programs. The various appendices in Section 7 describe important details about system organization and operation. Diagnostics are described in detail in Section 7.2 and a command summary is included in Section 7.4.

The BNF meta-language is used on some places to describe the language syntax. It is hoped that those not familiar with BNF will find the associated word descriptions adequate for understanding the syntactic descriptions and learning BNF.

A new user who wants to learn the language rapidly can probably benefit most from skimming Sections 2 and 3, reading Section 4 and then trying the system (get on the teletype) using Section 5 to learn the loading procedure and referring to the command summary for further details.

2. Data Structures

The basic data element in the ALGEBRA language is the algebraic monomial: a product of constants or parameters (collectively termed the "coefficient") and variables with associated exponents. Using standard algebraic notation a typical monomial might be written as

$$+23X^2Y^3Z$$

This expression may be considered a monomial in the variables X, Y, Z, a monomial in Y, Z with X as a parameter, or any other mathematically acceptable combination.* If the variables are X, Y, Z then +23 is the coefficient while if the variables are X, Z

---

*Note that the use of the word "variable" here is rather tricky. The X, Y are not variables in the FORTRAN sense since the monomial is data in the form of a string of symbols and X, Y, Z are constants in that data string. However, they are variables in the monomial. To be very careful, one should probably define data variables and language variables. However, since we would rather avoid this pedantry, the word variable will be used without modifiers when the meaning is clear from context.

then $+23Y^3$ is the coefficient. In ALGEBRA I data variables and parameters must be represented by single letters and it is usually necessary to make an explicit statement about which letters are variables and which are parameters. From monomials we can construct polynomials and rational function (ratios of polynomials) by the usual rules of algebra [7]. The purpose of the ALGEBRA system is to provide a tool for complex manipulations on the data-monomials, polynomials and rational functions.

Ideally one would want the set of all possible data to be closed under the set of allowable operations. This is the case for example in the FORTRAN language where the set of data is the field of real numbers: a set that is closed under the operations of addition and multiplication. In the development of an algebra manipulation system a similar result is obtained if the data is the field of rational functions with real coefficients and the operations are addition and multiplication. This set is closed under these operations and it remains so if one allows the additional operations of division, differentiation, and substitution of a rational function for a variable. However, when one begins to allow literal expressions including parameters in the coefficients and exponents of the polynomials, practical limitations on the complexity of the method of internal representation of the data causes a breakdown in the desired closure properties. An analogous breakdown in closure appears in numerical manipulation systems,

(e.g., FORTRAN, ALGOL) as a limitation on the size of the data elements (numbers) which can be handled. In the ALGEBRA I system this breakdown in closure appears as a limitation on the complexity of literal expressions appearing in the data (polynomials and rational functions) that is accepted as input or obtained as the result of manipulation. This section gives a complete discussion of the data limitations in ALGEBRA I. Unfortunately these limitations are somewhat complex and not always as easy to remember as the parallel limitations appearing in numerical languages. However, the user of simple polynomials with few parameters can be reasonably certain that he will encounter no difficulty. The user of more complex structures can either study this section in detail or let the system diagnostics "teach" him the current data limitations. Hopefully the built in tests on allowable data format will keep him out of trouble.

Before proceeding it is worthwhile to point out one other data format limitation occurring because of the physical limitations of the teletype, which is currently the only available input/output device. The monomial $+23X^2Y^3Z$ discussed earlier would be expressed to the ALGEBRA system on the teletype as

+23X↑2Y↑3Z

Any polynomial or rational function would have all of its monomials expressed in an equivalent fashion as exponents must always be indicated by ↑.

Since polynomials and monomials are special cases of rational functions, the term RF will be used in the sequel to represent the generic class of data-objects - monomials, polynomials, and rational functions - treated by the ALGEBRA system. The term rational function will always denote a non-degenerate rational function (i.e., a ratio of polynomials).

2.1 Expressions

In the formation of monomials certain groups of terms called expressions may be used. One kind of expression will be called a coefficient expression. A coefficient expression is a string of characters inside of parentheses and possibly having a numeric exponent. Examples are:

```
(A+B+3X↑B)↑2
(COS(X+3)+3(X+Y))↑3
(5(X+Y)+3X↑(A↑2+4))
(A+3XY↑(5X+3))
(A+B+(5X+Y)/(3X+Z))
```

A coefficient expression may appear as a part of the coefficient in a monomial. It is said to be convertible with respect to a given set of variables* if the contents of the outer set of parentheses is an RF in that set of variables conforming to the RF

*By this we mean letters which are understood to be variables in the context in which this expression appears (see Section 3.1).

format limitations to be given below. In the examples given above only the first, second and last expressions are convertible if X, Y are variables.*

A second kind of expression used in monomial formation is the exponent expression. The exponent expression is a polynomial with no exponents, variables or parentheses included in an outer set of parentheses. Examples are:

(-2A+3)

(4A+BC)

(-3C)

where A, B and C are not variables in the RF where these expressions appear.

2.2 Monomials

The basic form for a simple monomial is

(coefficient)(variable)↑(exponent)

Within the ALGEBRA I system the following restrictions on the data format of simple monomials must be observed:

---

*The COS(X+3) term in the second expression will be treated simply as a product of the three parameters C, O, and S time the coefficient expression (X+3). The ALGEBRA I system does not recognize transcendental functions.

1. Coefficients - may be numbers, parameters* or coefficient expressions. Numbers may be in FORTRAN I or F formats. Parameters may have exponents but the exponents must be numeric. Coefficient expressions may have exponents as described in 3 below.
2. Variables - single letters specified as variables by the user. (The specification procedure is described in Section 3.1 below.)
3. Exponents - an exponent expression, or a monomial of numbers or letters not containing any exponents (the + sign may be omitted in this monomial), or variables.

An ALGEBRA monomial will now be defined as a signed concatenation of simple monomials whose component parts satisfy the restrictions stated above. Thus

⟨ALGEBRA monomial⟩:: = ⟨sign⟩⟨simple monomial⟩|⟨ALGEBRA monomial⟩⟨simple monomial⟩

where ⟨sign⟩ is either a + or a - and the restrictions 1 through 3 above are observed. Unity coefficients and exponents may be omitted and in the latter case the "↑" is also omitted.

Because of the physical limitations of the teletype some freedom available in mathematical notations is not available in ALGEBRA. For example both

*Parameters are single letters which are not understood to be variables in the context in which they appear (see Section 3.1).

$+2X^{2}3Y$ and $+2X^{23}Y$

might be translated into the form

+2X↑23Y

at the teletype. To avoid this sort of ambiguity in the description of monomials, all characters following an ↑, up to and including the first) or up to but not including the next variable are considered to be part of the exponent of the preceding variable. Thus if X, Y are variables

+2X↑23Y ~ $+2X^{23}Y$

while

+2X↑(2)3Y ~ $+2X^{2}3Y = 6X^{2}Y$

+2X↑(-3A+4)Y ~ $+2X^{-3A+4}Y$

+2X↑-3A4Y ~ $+2X^{-12A}Y$

2.3 Literal Polynomials and Rational Functions

A literal polynomial is simply a concatenation of ALGEBRA monomials.

⟨literal polynomial⟩::=⟨ALGEBRA monomial⟩|⟨literal polynomial⟩⟨ALGEBRA monomial⟩

Leading + signs can be omitted, if desired. The variables in a literal polynomial are specified at the time of input as described in Section 3.1. All other letters are assumed to be parameters. Coefficient expressions appearing in a literal polynomial are extracted by the algebra system and treated as separate terms during most manipulation. In this way, expressions appearing in

the input may be preserved in the output of many manipulations. Examples of literal polynomials are:

3X↑2+2XY+5(A+3K)↑2Y↑2

3(A/B+9)XW↑(-2B+3)+7X↑2W+3

Literal rational functions are simply ratios of literal polynomials.

⟨literal rational function⟩::=(⟨literal polynomial⟩)/(⟨literal polynomial⟩

When the numerator or denominator polynomials of a rational function are simply unsigned monomials the parentheses may be omitted. Examples of literal rational functions are:

(2X↑2+3)/(-27XY+3Z)

(A+3B↑2)/X

((2A+B)↑2X+4)/(5AX+6)

In keeping with the previously mentioned policy of using the term RF to describe monomials, polynomials, or rational functions we now define

⟨literal RF⟩::=⟨literal polynomial⟩|⟨literal rational function⟩

3. The ALGEBRA Language

Statements in the ALGEBRA language are used to assign names to RF and to describe manipulations to be performed upon the RF's which are known to the system.

3.1 RF Names

In the description of manipulations on RF it is usually convenient to use names rather than literal statements of RF value.

Names of the form Pn, where n is an unsigned integer, may be assigned to RF by simple assignment statements of the form

⟨name⟩ = ⟨literal RF⟩

An example of a simple assignment statement is

P2 = 3X↑2+4XY+5

This statement assigns the name P2 to the literal RF appearing on the right of the equal sign.

At the time the RF is assigned a name its set of variables is also defined. The set of variables for a specific RF may be assigned both implicitly and explicitly. An explicit variable assignment is made by including an explicit variable list in parentheses after the name in the left hand side of the assignment statement. The assignment statement format is then

⟨name⟩(⟨explicit variable list⟩) = ⟨literal RF⟩

Thus the assignment statement

P2(X,Y) = 3X↑2+4XY+5

defines P2 as the literal polynomial on the right with X, Y as its explicit variable list. Letters appearing in the explicit variable list are the <u>explicitly assigned variables</u> of the RF.

All letters which have been explicitly assigned as variables in ALGEBRA statements are stored by the ALGEBRA system in a global variable list called VAR. When a new RF name with an explicit variable list is input, its explicit variable list is merged with

VAR so that at any time VAR contains all characters which have been defined as variables since the system was initialized.* When an assignment statement does not contain an explicit variable list all letters appearing in the literal RF which are currently on VAR are taken as variables for the RF. These letters are said to be implicitly assigned variables of the RF.

Letters appearing in the literal RF part of an assignment statement which are not on VAR or included in an explicit variable list following the name are considered as parameters and are stored by the ALGEBRA system in the global parameter list** which is called PAR. If at any time a parameter (a letter contained on PAR) is included in an explicit variable list, the warning message

PARAMETER USED AS VARIABLE

is printed to the user and the statement containing the erroneous explicit variable list is rejected.

In general an RF may have both implicitly and explicitly assigned variables. That is, some letters appearing an RF may be explicitly assigned as variables because they appear in an explicit

---

*The user may examine the contents of VAR with the command VARIABLES. See Section 4.2 below.

**The system name for this string is PAR but the user may examine it with the command CONSTANTS. See Section 4.2 below.

variable list while other letters may be implicitly assigned as variables because they appear on VAR at the time the RF is named. For example, if VAR contains X, Y then the assignment statement

P23(X,W) = (3X↑B+2W)/(5W+4W↑3+3Y)

defines P23 as the name of the literal rational function on the right with X, W as an explicit variable list. Following acceptance of this statement by the system, the string VAR contains X, Y, W and the letter B has been added to PAR. In this case the letters X, W are explicitly defined variables of P23 and Y is an implicitly defined variable.

The list of all variables appearing, both explicitly and implicitly, in a literal RF is termed the <u>local variable list</u> for the RF. In the above example the local variable list for P23 is X, Y, W. This list is used in the generation of the internal representation of the RF (Section 7.1). Moreover, all monomials in the RF must satisfy the format restrictions of Section 2.2 with respect to its own local variable list. Thus if VAR contains W, X, Y, Z then the statement

P11(X,Y) = 3X↑W+2XY

would have X, Y, W as a local variable list and therefore has an illegal format (variable in exponent). <u>Note that the letters P, D, I may not be used as data variables or parameters in the ALGEBRA I system</u>.

3.2 ALGEBRA Expressions and Manipulation Statements

An ALGEBRA expression is a mathematically correct combination of RF, parentheses and operators. The operators which can appear in ALGEBRA expressions are:

+	addition
-	subtraction
*	multiplication
/	division
//	numerical division
D( )/DX	differentiation
I( )DX	integration

Details about the operators and their operands are given in the next section.

Since RF can be assigned names it is usually more convenient to use RF names rather than literal RF in forming ALGEBRA expressions. Thus in discussing ALGEBRA expressions the term RF will be assumed to mean either RF names or literal RF. When an RF name appears in an expression the inclusion of an explicit variable list is optional. If one is included the ALGEBRA I system checks it against the explicit variable list given when the RF name was assigned a value (i.e., when the RF was defined). If the two do not agree a substitution is indicated as discussed in Section 3.3 below. If no substitution is intended the inclusion of explicit variable lists

with names in expressions is acceptable but not recommended because of the inherent typing and system checking delays. Examples of ALGEBRA expressions are:

```
P2+P3-P10+P4(X,Y)

(P5-P7)*P8

2X↑2-3X+P5(2X+3)
```

ALGEBRA expressions are used to describe manipulations to be performed on literal RF and RF names. When RF are substituted for the names appearing in an expression the prescribed manipulation can be carried out and a result obtained. This result is termed the value of the expression. The results of the manipulations described by an ALGEBRA expression can be assigned a name with a manipulation statement. The general form for a manipulation statement is

```
⟨name⟩=⟨expression⟩
```

This statement assigns the value of the expression (i.e., the result of the manipulation specified in the expression) to the name on the left. For example

```
P3(X)=3X↑2+2X+5
```

followed by

```
P4=P3-(7X+100)
```

would give P4 the value

```
3X↑2-5X-95
```

3.3 ALGEBRA Operations

The operations available in ALGEBRA are basically the same as those available in elementary calculus. First, there are the four basic binary operators

addition	+
subtraction	-
multiplication	*
division	/

The arguments of these operators can be arbitrary ALGEBRA expressions. Division of two polynomials simply converts them to a rational function while the operations +, -, or * on rational functions leads to the generation of new rational functions over a single common denominator.

An additional binary operator, denoted by //, causes a numerical division of the coefficients in an expression. For example

```
P3=⟨expression⟩//5
```

would assign to P3 the value of the expression with all coefficients divided by 5.

In addition to the above binary operators, there are unary + and - operators and unary operators for the differentiation of RF and integration of polynomials. Differentiation is indicated by

D(u)/Dv

and integration is indicated by

I(u)Dv

where u is an expression and v is a letter on the global variable list.* In integration the expression u must have as its value a polynomial in which the variable of integration appears with only numeric exponents. These numeric exponents may have a zero value but may not be equal to -1. In addition, the variable of integration may not appear in any coefficient expressions.

An additional operation available in ALGEBRA is substitution. If Pn was originally defined with an explicit variable list then

Pn(⟨expression1⟩,⟨expression2⟩,...,⟨expressionk⟩)

has as value the new RF generated by substituting the indicated expressions for the explicitly defined variables in Pn. Substitution is made for the variables in the order in which they were given in the original explicit local variable list. Substitutions of this form cannot be used for implicitly defined variables or parameters (however, see WITH modifiers - Section 4.13). Substitutions producing results which violate the data format restrictions described in Section 2.2 or in cases where Pn contains non-convertible coefficient expressions are illegal and are rejected by the ALGEBRA system. A substitution where the number of expressions included in the substitution operation

*The parenthesis may be omitted if u is a single name or monomial.

(i.e., the number k in the example Pn above) is less than the number of variables in the original explicit variable list is an illegal null substitution. In the opposite case where there are too many expressions to be substituted the excess expressions are ignored and the substitution is excepted. An appropriate diagnostic indicates what type of illegal substitution has been attempted (see Section 7.2).

Some examples of acceptable manipulation statements using the above operators are:

```
P1=5X+D(P3*P4+P5)/DX

P3(X,Y)=((P4+5XY)*P7)//7

P8=D(P4+I(5YZ*P3)DY)/DX

P9=P4+I(P3(P4,P3))DY
```

The usual precedence rules for operators in algebra apply. The numerical division has the same precedence as division and successive divisions are performed from right to left. That is

```
P1=P4/P3/P2/P5
```

means

```
P1=P4/(P3/(P2/P5))
```

Parentheses, ( ) and [ ], may be used freely to indicate sequencing. Parentheses surrounding literal RF appearing in ALGEBRA expressions do not lead to the introduction or coefficient expressions in the RF involved. For example in

P1=P5-(7X+3(A+Z)Y)

the term 7X+3(A+Z)Y is not treated as a coefficient expression but the term A+Z is treated as a coefficient expression. To avoid confusion all multiplications in ALGEBRA expressions involving anything more complex than constants and variables must be expressed by the * operator.

3.4 Output

A manipulation statement containing an expression on the left hand side of an equal sign and null on the right hand side is interpreted as a request for output of the value of the expression on the left. In this case the value of the expression on the left is computed and printed on the current output medium (usually the teletype) but no naming of this value takes place. The output is normally preceded by a copy of the expression on the left. For example, if

P1=2X+3

P2=4X↑2+2X

then the manipulation statement

P1=

would produce the output

P1=2X+3

the manipulation statement

2*P1+P2=

would produce the output

```
2*P1+P2=4X↑2+6X+6
```

and

```
P1+3XY=
```

would produce the output

```
P1+3XY=2X+3+3XY
```

It is sometimes desirable to suppress the copy of the request so that only the value requested is printed. This can be indicated by enclosing the output request in square brackets. Thus the manipulation statement

```
[P1] =
```

would produce the output

```
2X+3
```

and

```
[P1+P2]=
```

would produce the output

```
4X↑2+4X+3
```

3.5 Indirect RF Names

The RF names mentioned previously (Section 3.1) were limited to the form Pn where n was a positive integer. More general RF names may be used in the form

```
P[⟨expression⟩]
```

where the value of the expression is used to determine the RF name. Thus

```
P[I]=1
P[J]=2
P[P[I],P[J]](X,Y)=3X+4Y↑2
```

may be used to generate subscripted RF names.

In another example, if

```
P1=2X↑2+3XY+4Y↑2
P2=2
P3=5
P4=2*P2+P3
```

then

```
P[P4]=27X↑2+5Y
```

would assign the name P9 to the literal RF on the right. Similarly the output request

```
[P[P3-2*P2]]=
```

would produce the output

```
2X↑2+3XY+4Y↑2
```

Indirect RF names may appear anywhere where RF names are legal.

4. The ALGEBRA System

The ALGEBRA system must be loaded under the current version of SNOBOL 67. A description of the structure of ALGEBRA I and the SNOBOL subroutines used in the ALGEBRA system is given in Section 7.3. A precompiled version of the ALGEBRA I system created with the STORE command in SNOBOL 67 is usually available and can be loaded and run without compilation. Loading procedure is described in Section 5.1.

When completely loaded the ALGEBRA system responds with a short version message and then spaces over 4 spaces and prints a \ in column 5 indicating it is ready for user input. The user may then type ALGEBRA Statements or Commands.

4.1 ALGEBRA Statements

The basic format for an ALGEBRA Statement is

⟨label⟩:⟨modifier⟩⟨manipulation statement⟩;⟨goto⟩

where the label is an arbitrary string of characters (except colons) followed by a colon, the modifier is one of the modifiers to be described below (Section 4.1.2), the manipulation statement describes any of the legal manipulations covered in Section 3 and the goto is a label preceded by a semicolon. Blanks may be added arbitrarily and the statement is terminated with a carriage return (CR). One or more of the elements of a statement may be omitted. Some example statements are:*

```
\P1(X,Y)=3X↑2Y+2 cr
\MP1:WITH Y=3!P3=P1*P1;MP10 cr
\AX1:;AXØ cr
\;BQ1 cr
\ABC:P4=5*P1+3*P2 cr
\AX2:P4+P1= cr
```

4.1.1 Statement storage and execution

Statements without labels and not preceded by labeled statements are not stored by the ALGEBRA system. In the normal operation

*The \ is included for clarity in the examples although not typed by the user. The cr indicates a terminating carriage return typed by the user.

(execute mode) such statements are executed immediately after receipt and then discarded. If they require output the output is given immediately after execution is completed. Once a label is encountered the ALGEBRA system starts saving input statements. All future statements are then saved (even unlabeled statements) until an END statement is encountered.* After an END statement unlabeled statements are again discarded until a label is encountered.

Independent of the statement saving features mentioned above is an execution control feature. In the normal execution mode all statements are executed immediately after input. The alternate procedure is the read mode in which statements may be saved without execution. In the read mode saved statements will not be executed until called as a procedure or executed via a TØ command (Section 4.2.2). The execute or read mode may be set with the R and X commands described in Section 4.2 below.

Combinations of the above options on statement saving and execution give four different modes of operation in the ALGEBRA I system:

Mode 1. Statements executed (mode X) and discarded - this mode is for the generation of simple results and non-repetitive operation.

*The meaning of the END statement is explained in the next section.

Mode 2. Statements saved and not executed (mode R) - this mode is for the construction of blocks of statements to be executed at a later time.

Mode 3. Statements executed (mode X) and saved - this mode is for the construction of blocks of statements for possible later execution and provides checking the results as the block is constructed.

Mode 4. Statements discarded and not executed (mode R) - this mode is of little value.

Modes 2 and 3 provide for the construction of blocks of statements to be executed as programs or procedures (subroutines). The use of these features is considered in the next section.

4.1.2 Direct statements, programs and procedures

Statements executed but not saved in mode 1 may be called direct statements. Examples of a set of direct statements might be and the resulting output* are:

```
\P1(X,Y)=2X+3Y↑2 cr
\P2=2XY+5 cr
\P2*DP1/DY= cr
 P2*DP1/DY=12XY↑2+30Y
```

Note that all statements are executed and output requests answered* immediately after the terminating CR. If the first of the above statements were labeled or a labeled statement had preceded them

*Response to output requests are started in column 1, while the ready symbol ("\") appears in column 5.

then these statements would still be executed immediately but they would also be saved so that they could be re-executed later.

An ALGEBRA program is a block of statements terminated by an END statement: a statement of the form

⟨label⟩:END

where the label is optional. It may be executed by using the TØ command (Section 4.2) or by execution of a direct statement containing a goto to a statement in the program. An ALGEBRA program is roughly equivalent to a SNOBOL or FORTRAN program. Saved statements can be executed, deleted, edited, appended to, or printed on the teletype by using the commands given below (Section 4.2). A program may be created in mode 2 or 3 of Section 4.1.1. The latter mode allows the user to test the program by examining the intermediate results of one test case as the program is being written. As an example of a program the saved statements

```
\B1:P1(X)=3X↑2+2X+9
\DP1/DX=
\END
```

could be saved in mode 2 or 3 and later executed with the statement

```
\TØ B1
```

Execution would terminate with the END statement even though other saved statements followed. If an END statement is not encountered execution terminates with the last saved statement (assuming it does not contain a goto).

Saved statements may also form a procedure, roughly equivalent to a function in SNOBOL or FORTRAN. A procedure is a block of ALGEBRA statements terminated by a special END statement of the form

⟨label⟩:END PRn(P1,...,Pm)=Pk;⟨entry label⟩

where the label is optional. The procedure name is PRn; where n is an integer, P1 through Pm are dummy RF names of the arguments, and Pk is the name of the RF whose value is assigned to the name of the procedure when the procedure returns. The entry label is the label of the entry point for the procedure. When a procedure is called by the appearance of a procedure name in an expression, current values of the dummy names used in the definition are saved. Then any expressions appearing as arguments in the calling statement are evaluated and assigned to the dummy names used in the definition. Execution then begins at the entry point and continues until an END statement (of either type) is reached. At this point the RF indicated in the definition (Pk in above) is evaluated and its value assigned as the value of the procedure name. The global values of the dummy names are then restored and control is returned to the calling ALGEBRA statement for further execution. The entry and exit processes are structured so that procedures can be recursively defined, without any specific statements by the user that he is making a recursion definition.

Procedure calls may occur in ALGEBRA statements in the same manner as polynomial names and may have arbitrary ALGEBRA expressions as arguments. In all operations procedure calls are treated as RF names and the value returned to the procedure name is used in obtaining the value of any expression where the procedure call appears.

As an example, the procedure

```
\SP:P1(x)=3X+4X↑2
\P2=D(P1+P4*P5)/DX
\END PR1(P4,P5)=P2;SP
```

would assign the value of P2 to the name PR1 when it was executed. It might be called in an ALGEBRA statement of the form

```
\P12=P10+P11*PR1(P8,P9+5*DP10/DX)
```

or it could call itself as in

```
\P12=P10+PR1(PR1(P8,P9),P11+P9)
```

4.1.3 Statement modifiers

Three types of statement modifiers are available in the ALGEBRA I system. The WITH modifier is used to provide for substitution for variables and/or parameters before the manipulation is carried out. The SNOBOL modifier is used to test results obtained, provide conditional branching in ALGEBRA programs or call user defined SNOBOL programs. The COMMENT modifier can be used to print comments or titles in the output data. Any one or all three types of modifiers may appear in a single ALGEBRA statement.

The basic format for the WITH modifier is described as follows:

```
⟨letter list⟩::=⟨letter⟩|⟨letter list⟩,⟨letter⟩
⟨expression list⟩::=⟨expression⟩|⟨expression list⟩,⟨expression⟩

⟨WITH modifier⟩::=WITH ⟨letter list⟩=⟨expression list⟩!|
     ⟨WITH modifier⟩⟨letter list⟩=⟨expression list⟩!|
     ⟨WITH modifier⟩WITH ⟨letter list⟩=⟨expression list⟩!
```

A typical example of a simple WITH modifier might be

```
WITH B=10!C=5!
```

or the equivalent form

```
WITH B,C=10,5!
```

This modifier might be used in the statements

```
\L1:P1(X)=3X↑2+2BX+C
\WITH B=10!C=5!P1=
```

which would result in the output

```
P1=3X↑2+2OX+5
```

In most cases when multiple substitutions are prescribed in one WITH statement the substitutions are made simultaneously. This means that

```
WITH B=C!C=A!
```

or equivalently

```
WITH B,C=C,A!
```

will replace all B's with C's at the same time that it is replacing all C's with A. The resulting statement will contain both C's and A's. This mode of substitution is in opposition to a sequential mode where all B's are replaced with C's and then all C's are

replaced with A's. The result would then not contain any C's since the last step replaces all C's with A's. While most substitutions made with the WITH modifier are simultaneous, limitations on the manipulation routines in ALGEBRA I leads to the occurrence of some sequential replacements. The specific rules for substitution are as follows:

1. Simultaneous substitution for all variables appearing anywhere except in non-convertible coefficient expressions (defined in Section 2.2).
2. Simultaneous replacement for all parameters except sequential replacement for parameters appearing in exponent expressions.
3. No substitutions are made into non-convertible coefficient expressions.

When a non-convertible coefficient expression is detected during an attempted substitution the substitution is interrupted and a message indicating the conversion difficulties is output to the user.

Multiple occurrences of the word WITH in the modifier can be used to force sequential replacement operations. In this case the right most WITH is completed first with execution of other WITH's proceeding to the left. Thus the modifiers

```
WITH C=A!WITH B=C!
```

appearing in one statement would first cause B's to be replaced by C's and then C's to be replaced by A's regardless of where the B's and C's occurred (as long as they were not contained in non-convertible coefficient expressions).

When WITH modifiers are used in statements containing procedures the WITH is applied to the value returned by the procedure before the manipulation is executed. However, the WITH is not applied to the procedure arguments before the procedure is evaluated. Thus with PR1 defined by

```
\L1:P1=DP3/DX
\P2=P1+3Z↑2
\END PR1(P3)=P2;L1
```

the statement

```
\WITH X=Z!PR1(3X↑2)=
```

produces the output

```
6Z+3Z↑2
```

WITH statements can be used to set the values of indirect RF names if desired. Thus

```
\WITH J=2!P[J]=
```

outputs the value of P2 when executed.

A SNOBOL modifier allows one or more SNOBOL statements to precede the manipulation.* The modifier format is

```
⟨SNOBOL modifier⟩::=⟨SNOBOL statement⟩?|
      ⟨SNOBOL modifier⟩⟨SNOBOL statement⟩?
```

*A familiarity with SNOBOL 67 is required to understand parts of the remainder of this discussion. See [1].

The ALGEBRA statement in which this modifier appears and any goto appearing in that statement will be executed only if all the SNOBOL statements in the modifier succeed. Failure of any one of the SNOBOL statements will cause an immediate transfer to the next (sequential) ALGEBRA statement in the program or procedure being executed.* SNOBOL modifiers are quite useful in testing intermediate results obtained in a program or procedure. The results of such tests can be used with gotos to provide conditional branching in ALGEBRA programs.

Some special SNOBOL functions which can be called in a SNOBOL modifier to examine values obtained in ALGEBRA statements are:

V('u') returns the value in ALGEBRA internal form of the ALGEBRA expression, u. (See Section 7.1 for a discussion of ALGEBRA internal form.)

N('v') succeeds if v is a RF in internal form which is numeric. Returns this numeric value to N.

Note that single quotes are needed around the arguments of the functions N and V since these are literal strings to the SNOBOL system. Other SNOBOL functions defined by the user can also be called in SNOBOL modifiers. An example illustrating the use of a SNOBOL modifier or an ALGEBRA statement might be

---

*The SNOBOL statements should not have gotos as this would result in a transfer out of the ALGEBRA system. This type of error will cause a diagnostic message from the SNØBØL compiler.

```
\P1=1
\LOOP1:P2=3*(P4+P1)*P5
   :
   :
\.EQ(N(V('P1')),'10')?;OUT
\;LOOP1
\OUT:···
```

Here P1 is used as a loop counter. When P1=10, a transfer out of the loop (to the ALGEBRA statement labeled OUT) is executed. In another example, a loop containing

```
\V('P1*P3')  'X3Y5'?P1*P3=
```

would cause a printout of the value of P1 times P3 whenever the product contained a term of the form

```
X↑3Y↑5
```

A COMMENT modifier prints a comment in the output when the statement in which it appears is executed. The modifier format is

```
⟨COMMENT modifier⟩::="⟨character string⟩"|
     ⟨COMMENT modifier⟩"⟨character string⟩"
```

When a statement in which a comment modifier appears is executed the character string appearing in the double quotes is printed before the manipulation statement is executed. Thus an output requested in the manipulation statement will appear after the comment. For example, when executed the lines

```
\P1(X)=3X↑2+4X+5
\L1:"THE VALUE OF P1=" [P1]=
```

will produce as output

```
THE VALUE OF P1=3X↑2+4X+5
```

with the T of THE located in column 1.

WITH, SNOBOL and COMMENT modifiers can be mixed in an arbitrary fashion in the development of a general modifier. Thus

⟨modifier⟩::=⟨WITH modifier⟩|⟨SNOBOL modifier⟩|⟨COMMENT modifier⟩|
⟨modifier⟩⟨modifier⟩

is the general modifier that can appear in an ALGEBRA I statement.

4.1.4 The PAUSE Statement

A special type of ALGEBRA statement which may appear in any program is the PAUSE statement. It may have a label or modifier but no goto. Thus the format for a PAUSE statement is

⟨label⟩:⟨modifier⟩PAUSE

When this statement is executed the system prints the message

PAUSE AT p

where p is the label of the PAUSE statement or a preceding label and a displacement, the system then returns to the ready symbol \. At this point, the user may examine values with direct statements, change values, etc. Execution may be restarted at the statement following the PAUSE, or any other statement, by using the TØ or START commands (Section 4.4.2).

4.2 System Commands

System Commands in ALGEBRA I provide the user with a capability for control of system operation mode, execution of programs, statement and program editing, program storage and program retrieval.

The commands are described below by word name but in all cases only the first letter of the word need be typed--all remaining characters are ignored. Commands may be issued any time the \ has been printed to indicate the system's readiness for input. Certain commands require arguments, as described below, and all commands (with possible arguments) must be terminated with a CR. Lines containing system commands are not saved in any system mode (Section 4.1.1).

4.2.1 Statement reference

Statement references required in execution commands and editing commands may be given in any one of three forms:

1. By statement number. Statements are numbered sequentially in the order in which they are saved. The first saved statement is numbered "1".
2. By label, or label plus positive integer displacement.
3. By "$", which denotes the last line that has been saved.

Since not all lines are saved one must be careful to obtain the correct line. The PRINT command given below (Section 4.2.3) may be used to examine a line when the user is in doubt.

Some examples of statement references are:

L1	statement with label L1.
L1+5	fifth statement after statement labeled L1.
2	the second statement saved.
$	the last statement saved.

4.2.2 Mode and execution control commands

The system execution mode (Section 4.1.1) may be modified with the commands:

READ — accepts ALGEBRA statement input without immediate execution

X — all input ALGEBRA statements are executed immediately (normal mode)

The following commands may be used to execute saved blocks of statements or to continue execution following a PAUSE. The symbol p represents a reference to a saved statement in any of the three forms described in Section 4.2.1.

TØ p — begins execution of an ALGEBRA program starting at statement p. This can also be used to restart a program after a PAUSE is executed not inside a procedure.

START p — used to restart after a PAUSE has been executed inside a procedure.

4.2.3 Editing Commands

The simplest editing commands are the character and line editing commands used to correct errors in input. (A^C or Q^C means hit the A or Q while depressing the control key on the teletype.)

A^C — deletes the preceding character and prints ↑. May be repeated an arbitrary number of times to delete a string of preceding characters.

Example: \P1+?↑P2...
A^C

Typing A^C after the ? causes ? to be deleted and ↑ to be printed. Input then proceeds as usual.

Q^C deletes the current line if it has not been terminated with a CR. Prints ← to indicate deletion and then causes a CR and line feed. The new version of the line is thus begun in column 1 (ALGEBRA input normally begins in column 5)

Example:
```
\P5(3,*←
P2(3,X)=
```
Typing Q^C after * prints the ←, deletes the line and causes a line feed and CR to column 1 where a new version of the line is begun.

Additional editing commands are used to modify already saved blocks of ALGEBRA statements (see Section 4.1.1 about saving statements). The general format for a statement editing command is

⟨command name⟩ ⟨statement reference1⟩,⟨statement reference2⟩

The comma can be replaced by a space if desired. When only one statement location is needed the comma and the second location can be omitted.

The statement editing commands are listed below. The p and q are statement references, each in one of the formats described in Section 4.2.1.

APPEND p prints the existing version of line p to which the user may append additional statement structures. The additional material is typed immediately following the printing of the existing version. The addition is terminated with a CR.

Example:
```
\A L1
L1:P3=P4+P2*P1
```
prints the line labeled L1 to which the user may add a goto, further manipulation, etc., terminated with a CR.

DELETE p,q deletes statement p through statement q. If q is missing, only statement p is deleted.

EDIT p	replaced statement p with the line typed below the command.
Example: \E L1 L1:P4=P8+P5;L2	replaces the original line labeled L1 with this line.
LABEL p,⟨label⟩	puts the given label on statement p
Example: \L L1,Q2	changes label on statement L1 from L1 to Q2.
\L L1+5,Q3	puts label Q3 on statement fifth statement after L1
PRINT p,q	prints statements p through q on teletype (starting in column one). If q is missing prints line p.

4.2.4 Miscellaneous commands

CONSTANTS	causes printing of the word PARAMETERS: followed by the current parameter list.
VARIABLES	causes printing of the word VARIABLE: followed by the current variable list.
WRITE p,q ON /file/	causes ALGEBRA statements p through q to be written onto the disc in the indicated file (ON is printed by the system)
READ /file/	reads ALGEBRA statements from the indicated file. Appends statements from the file to existing saved statements.
Examples: \WRITE L1,L1+5 ON /ALGP10/	writes ALGEBRA statements L1 through L1+5 on disc file /ALGP10/.
\READ /ALGP10/	reads ALGEBRA statements from disc file /ALGP10/.

5. Sample Programs

This section provides some sample programs illustrating the application of ALGEBRA I. Most of the features of the language

will be illustrated, but due to space limitations the applications discussed will be rather elementary.

5.1 Loading Procedure

The ALGEBRA I system is loaded under SNOBOL 67, a special version of COM-SHARE SNOBOL. It is here assumed that we have already "logged in" to the COM-SHARE system and are communicating with the COM-SHARE Executive. The necessary statements for loading SNOBOL are as follows (statements typed by the user are underlined--each user input is assumed to be terminated with a carriage return--the cr for carriage return is only indicated in the examples where its use as a line terminator might not be obvious.

```
-EXE 254SNØ/SNØBØL/

SNOBOL 67 VER.6-3-69
NUMBER OF LITERAL STRINGS TO BE USED = cr

$READ
FROM 254SNØ /ALGEBRA/

$:T:T

ALGEBRA I
VER. XXXXX

   \
```

The - in the first line is the COM-SHARE Executive ready symbol. The first line instructs the Executive to load and execute the disc file containing the SNOBOL 67 program. The heading for SNOBOL 67 then appears, followed by the SNOBOL 67 ready symbol $. The READ tells the SNOBOL Executive to load the ALGEBRA I system and the :T:T begins the execution of the ALGEBRA program. The

heading for ALGEBRA I then appears, followed by the backslash, \, in column 5, indicating that the ALGEBRA I system is now ready to accept statements or commands as described in Section 4 above.

5.2 Simple Operations with Direct Statements

The following lines illustrate some direct statement manipulations. Note that all statements are typed after the \ in column 5, while output from the computer starts in column 1. An exception to this rule is the case where a line is deleted with Q^C. The deletion is indicated by a ← followed by a carriage return and line feed. The new version of the line is then started in column 1.

Here we start direct statements at the \.

```
     \P1(X,Y)=X+2Y↑2
     \P2(X)=3X↑2
     \P3=P1+3*DP2/DX
     \P3=
P3=19X + 2Y↑2
     \P4(X,Y,Z)=X↑2+XY
     \WITH X=EZ+FZ↑2!P4=
P4=E↑2Z↑2 + 2EFZ↑3 + F↑2Z↑4 + EYZ + FYZ↑2
     \P5=IP4DX-P3+P2*P1+100
     \P5=
P5=100 + 3.333333333X↑3 + 6X↑2Y↑2 - 19X - 2Y↑2 + 0.5X↑2Y
     \IP3DY=
IP3DY=19XY + 0.6666666667Y↑3
     \"THIS IS THE P4 PØLYNØMIAL  "[P4]=
THIS IS THE P4 PØLYNØMIAL    X↑2 + XY
     \P(2,?←                 (← due to a Q^C)
P1(2,5)=                     (correct version)
P1(2,5)=52
     \P6=P1-IP4DY+3X*P2-DP5/DY
     \P6=
P6=- 13X↑2Y - 0.5X↑2 + 4Y + 9X↑3 - 0.5XY↑2 + X + 2Y↑2
     \                       (type ESC followed by CR)
$                            (type G^C)
```

```
-LOG
:
:
TIME USED
CPU:    0.50 MIN
CON:    0.55 HOURS
```

Note that to leave the ALGEBRA I System you must do an ESC followed by a CR. This leaves you in SNOBOL 67 from which you exit to the COM-SHARE Executive with a G^C. Then type LOG (for log out) to the COM-SHARE Executive to disconnect from COM-SHARE. The time you used is given at log out.

5.3 Sample Use of a Procedure

Assuming that "log in" is completed, the following example illustrates the use of a procedure. This procedure generates Chebyshev polynomials using a standard recursion relation. The first part of this example is a listing (using the P1,$ command) of the procedure named PR1 which has been input and saved in earlier operations. The second part [starting at P19=PR1(5)] calls the procedure to print out Chebyshev polynomials up to number 5. Note how the procedure calls the N and V functions in a SNOBOL modifier to test whether the number of polynomials requested (stored in P100) is less than or equal to one.

```
    \P 1,$
CP1:"CHEBYSHEV PØLYNØMIALS UP TO NUMBER "[P100]=
\P1=1.
\P2=X
\P4=1
\.LE(N(V('P100')),'1')?;CP20
\[P1]=
\[P2]=
```

```
      \CP5:P3=2X*P2-P1
      \P1=P2
      \P4=P4+1
      \[P3]=
      \.EQ(N(V('P100')),N(V('P4')))?;CP30
      \P2=P3;CP5
      \CP20:[P[P100+1]]=
      \CP30:END PR1(P100)=P1;CP1
      \P19=PR1(5)
 CHEBYSHEV PØLYNØMIALS UP TO NUMBER 5
   1.
   X
 - 1. + 2X↑2
 - 3X + 4X↑3
 - 8X↑2 + 1. + 8X↑4
 - 20X↑3 + 5X + 16X↑5
```

5.4 Generation of a Symbolic Lyapunov Function

This is a rather complex example demonstrating the use of the ALGEBRA language in the generation of symbolic Lyapunov functions. The method of Lyapunov function generation illustrated is a modified form of Wall's procedure [8], one of the many generation techniques easily implemented in the ALGEBRA I system. This example is included only to illustrate that the language has some applicability to complex algebraic problems. It is not intended to suggest that the procedure illustrated is an efficient approach to Lyapunov function generation. Results relating to efficient algebra manipulation techniques for Lyapunov function generation will be published elsewhere.

This example uses a SNOBOL modifier which calls a rather complex, user coded, SNOBOL function called FACT. The call to FACT in this program simply returns to the RF name P199 the coefficient of the polynomial P2 found in the polynomial P3. The FACT

function is listed below following the loading and running of the Lyapunov generation program.

In this example the program is used to generate a Lyapunov function for the following third order system of ordinary differential equations:

$$\dot{x} = y$$

$$\dot{y} = z$$

$$\dot{z} = -y^2 z - z - y - x$$

the result is found to be

$$v(x,y,z) = \frac{1}{2} x^2 + xy + \frac{1}{2} y^2 + \frac{1}{2} z^2 + yz + \frac{1}{4} y^4$$

$$\dot{v} = -y^2 z^2$$

The program begins with user input of statements in READ mode. (Marking of user input with underlining is dropped in this example.) Following the END statement for PR1 the user switches to X mode and inputs the differential equation. He then uses a TØ command to start execution at the statement labeled VF. The output is the result of integrating certain functions of the right sides of the differential equations along special paths in x,y,z space. These outputs are tested (using procedures described elsewhere) for sign definiteness. The first output (denoted PATH XYZ and enclosed in brackets) is found to be positive definite and thus given a satisfactory Lyapunov function in this case.

```
\R
\VDT: P20=DP10/DX*P1+DP10/DY*P2+DP10/DZ*P3
\END PR2(P10)=P20;VDT
\VF: FACT(V('P3'),V('P2'))?
\P4(X,Y,Z)=-P3+P199*P2
\P5(X,Y,Z)=-P3-P199*P1
\P6(X,Y,Z)=P1+P2
\"W1 =    "[P4]=
\"W2 =    "[P5]=
\"W3 =    "[P6]=
\P12=IP4(X,0,0)DX+IP5(X,Y,0)DY+IP6DZ
\"PATH XYZ    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\P12=IP4(X,0,0)DX+IP6(X,0,Z)DZ+IP5DY
\"PATH XZY    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\P12=IP5(0,Y,0)DY+IP4(X,Y,0)DX+IP6DZ
\"PATH YXZ    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\P12=IP5(0,Y,0)DY+IP6(0,Y,Z)DZ+IP4DX
\"PATH YZX    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\P12=IP6(0,0,Z)DZ+IP4(X,0,Z)DX+IP5DY
\"PATH ZXY    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\P12=IP6(0,0,Z)DZ+IP5(0,Y,Z)DY+IP4DX
\"PATH ZYX    V =    "[P12]=
\"VDØT =    "[PR2(P12)]=
\END PR1(P97)=P4;VF
\X
\P1(X,Y,Z)=Y
\P2(X,Y,Z)=Z
\P3(X,Y,Z)=-Y↑2Z-Z-Y-X
\TØ VF
W1 =      Y + X
W2 =      Y↑3 + 2Y + Y↑2Z + Z + X
W3 =      Z + Y
SUCCESS {PATH XYZ    V =     0.5Z↑2 + YZ + 0.25Y↑4 + Y↑2 + XY + 0.5X↑2}
        {VDØT =     - Y↑2Z↑2                                          }
PATH XZY    V =     0.25Y↑4 + Y↑2 + 0.3333333333Y↑3Z + YZ + XY + 0.5Z↑2
 + 0.5X↑2
VDØT =     - 0.3333333333Y↑5Z - 0.333333333Y↑3Z - 1.018634066E↑(-10)Y↑2
Z↑2 - 0.3333333333Y↑4 - 0.3333333333XY↑3
PATH YXZ    V =     0.5Z↑2 + YZ + XY + 0.5X↑2 + 0.25Y↑4 + Y↑2
VDØT =    - Y↑2Z↑2
PATH YZX    V =     XY + 0.5X↑2 + 0.5Z↑2 + YZ + 0.25Y↑4 + Y↑2
VDØT =    - Y↑2Z↑2
```

```
PATH ZXY   V =      0.25Y↑4 + Y↑2 + 0.3333333333Y↑3Z + YZ + XY + 0.5X↑2
 + 0.5Z↑2
VDØT =     - 0.3333333333Y↑5Z - 0.333333333Y↑3Z - 1.018634066E↑(-10)Y↑2
Z↑2 - 0.3333333333Y↑4 - 0.3333333333XY↑3
PATH ZYX   V =      XY + 0.5X↑2 + 0.25Y↑4 + Y↑2 + 0.3333333333Y↑3Z + YZ
 + 0.5Z↑2
VDØT =     - 0.3333333333Y↑5Z - 0.333333333Y↑3Z - 1.08634066E↑(-10)Y↑2
Z↑2 - 0.3333333333Y↑4 - 0.3333333333XY↑3
```

The user coded SNOBOL function FACT is listed below to illustrate the use of SNOBOL statements in complex tests and evaluations. The SNOBOL statements in FACT are simply loaded (and therefore compiled) in SNOBOL 67 after the ALGEBRA I system is loaded but before it is executed. (It is usually advisable to erase the SNOBOL symbolics to save space - see [1].) After FACT is loaded the ALGEBRA I system is executed (using $:T:T - see Section 5.2) and any calls to FACT occurring in a SNOBOL modifier in the ALGEBRA program are automatically recognized.

```
$SCF FACT(P,P1;C,A,B,T,T1);F1
F1 SAME(P,P1,'P','P1') /(F10)
F6 T1 *\#* /F(F9)S(F20)
F8 $'::+199:T'←T1 /(RETURN)
F9 $'::+199:T'←'::+0:' /(RETURN)
F10 P1 *;'+-'* *T1;VAR* *T* ':' =T1 ':'
 T1='::'
 P ':' *;'+-'* *P*
F11 P *A;VAR* *B* ':' *P* /F(F6)
 .EQUALS(B,T) /F(F11)
 T1 = T1 A ':' /(F11)
F20 P1 *;'+-'* *A;PAR ':'* *B* ':'
 P =
 T1=.NE(A,'1') NDVD(T1,A)
 .NULL(B) /S(F8)
F21 B *(A)* *C;PAR* *B* /F(F23)
 C .ANCHOR() '-'='+' /S(F22)
 C='-' C
F22 P=P A C /(F21)
F23 T1=MUL(T1,'::+1' P ':') /(F8)
```

Note that FACT calls the SNOBOL function SAME, NDVD, and MUL which are basic parts of the ALGEBRA I system. Their function is mentioned in Section 7.3.

6. References

1. Kain and Bailey, SNOBOL 67: Users reference manual, Edition 4. Department of Electrical Engineering, University of Minnesota, Minneapolis, Minnesota 55455, May 1, 1969.
2. Sammet and Bond, Introduction to FORMAC. IEEE Trans. on Electronic Computers, EC-13, 4 (1964).
3. Brown, Hyde and Tague, The ALPAK system for non-numerical algebra on a digital computer, BSTJ, 42 and 43 (1963).
4. McIlroy and Brown, The ALTRAN language for symbolic algebra on a digital computer, BSTJ,
5. Koehler and Eaton, The AMTRAN programming system. TR-792-8-292, Nortronics-Huntsville, Huntsville, Ala., Feb. 1968.
6. Martin, Symbolic mathematical laboratory. Ph.D. Thesis, M.I.T., January 1967.
7. Birkhoff and MacLane, A survey of modern algebra, MacMillan Co., New York, 1953.
8. Wall and Moe, An energy metric algorithm for the generation of Lyapunov functions. IEEE Trans. on Auto. Cont., AC-13, p. 121, Feb. 1968.

7. Appendices

7.1 Internal Representation of Polynomials, Rational Functions and RF Names

The external language with which the user communicates with the ALGEBRA I system has been chosen for maximum user convenience and similarity to hand written algebra notation. However, since such notation is not the most convenient form for the manipulations to be carried out by the computer a conversion from external to internal form is carried out. For literal RF the external to internal conversion is handled by a subroutine called CONEI while RF names are converted to internal form by the ALGEBRA I Executive (REXEC). Since the user may, in some cases, need to deal with the internal representation of RF or RF names--for example if he uses the V function (Section 4.1) or defines a new RF in a SNOBOL modifier--the following paragraphs describe the internal representations employed in ALGEBRA I. An appreciation of the internal representation of literal RF also helps to explain some of the limitations on data format given in Section 2.

The internal to external conversion of RF is handled by a subroutine called CONIE. Some scrambling of terms is inevitable but every attempt is made to collect like terms whenever possible so that the output format is relatively neat and similar to the results expected in hand manipulation.

7.1.2 Representation of monomial terms

A polynomial is represented as a string of monomial terms so in this discussion the major emphasis will be placed on internal representation of monomial terms.

The internal representation of a monomial term has the form

⟨sign⟩⟨coefficient⟩⟨variable or expression⟩⟨exponent⟩···

...⟨variable or expression⟩⟨exponent⟩:

For example if X, Y, Z are variables and A, B, C, are parameters:

3X↑2Y↑3Z is represented as +3X2Y3Z1:

-3X↑(2A+B)4YZ is represented as -12X2A+1BY1Z1:

If there are any parameters appearing in the coefficient they may have numeric exponents which are represented as above. For example:

3A↑2X↑2Y is represented as +3A2X2Y1:

Coefficient expressions are removed from the monomial and replaced by dummy names denoted by integers inside of exclamation marks, such as "!3!" Each integer corresponds to a different coefficient expression. For example:

-17A(2B+C)↑2X↑3 is represented as -17!1!2A1X3:

where the !1! represents the expression (2B+3) and the 2 indicates the fact that the expression is squared. (Note that exponent expressions <u>are not</u> replaced by dummy names but coefficient expressions are replaced by dummy names.) More examples are:

-23A↑2(3X+2)↑(2A+1)X↑(3A+2)Y(2Z+5X)

is represented as -23!2!2A+1!3!1A2X3A+2Y1:

where !2! represents 3X+2 and !3! represents 2Z+5X. Similarly,

+15A↑3X↑(27B-C)6BY↑-3Z↑(-2A)

is represented as +90A3B1X27B-1CY-3Z-2A:

7.1.3 Representation of polynomials

Since polynomials are simply strings of monomials, the representation is likewise a string of monomial representations. However there are two additional requirements. First, each polynomial carries with it a local variable list (see Section 3.1). This local variable list appears at the beginning of the internal representation. Second, each monomial in the polymomial must include all variables in the local variable list. Those not actually appearing in the external monomial representation are written in the internal form with a zero exponent. Variables in the internal representation of a polynomial always appear in each monomial in the same order in which they appear on the local variable list while the local variable list is ordered according to the sequence of occurrence of its variables in the global variable list VAR. This order corresponds to the order of their declarations as variables in statements as they are executed (Section 3.1).

The general form for internal representation of a polynomial is

:⟨variable list⟩:⟨monomial⟩⟨monomial⟩···⟨monomial⟩

If it is assumed that the global variable list VAR contains only X, Y then

P1(X,Y)=3WX↑2Y↑-3+2(A+B)↑2XY+27X+5

is represented as :XY:+3W1X2Y-3:+2!1!2X1Y1:+27X1YO:+5XOYO:

P2(X,Y)=3 is represented as :XY:+3XOYO:

but

P3=3 is represented as ::+3:

However if the global (system) variable list VAR contains XYWZ then

P4=27X+3Y is represented as :XY:+27X1YO:+3XOY1:

while

P5=27Z+3X is represented as :XZ:+27XOZ1:+3X1ZO:

Note that the letters on the local variable list always appear in the same order as they appear on the global variable list. However, manipulations can increase the size of the local variable list of any polynomial.

If VAR contains XYWZ then

P7(X,Y)=9(A+B)↑2B↑3X↑(2+B)Y+3W↑2X4Y+7

is represented as

:XYW:+9!2!2B3X2+1BY1WO:+12X1Y1W2:+7XOYOWO:

Here the W is included in the polynomial variable list because it is on the global variable list (i.e., it is an implicitly defined variable of P7).

7.1.4 Representation of Rational Functions

Since rational functions are ratios of polynomials, their internal representation is simply a modification of the representation of two polynomials. The general form for the internal representation of a rational function is

/⟨polynomial⟩/⟨polynomial⟩

There is an additional requirement that both polynomials in an internal rational function must have the same local variable list. Note that an input which is in mixed form externally such as

P1(X,Y)=X+Y/(3X+2)+4Y

is transformed to a ratio of two polynomials (put over a common denominator) before conversion is completed. Thus the internal representation of the P1 above would be the same as the internal representation of

P2(X,Y)=(3X↑2+2X+12XY+9Y)/(3X+2)

The common internal representation would be

/:XY:+3X2YO:+2X1YO:+12X1Y1:+9XOY1:/:XY:+3X1YO:+2XOYO:

It is important to note that mixed functions of apparently simple form may require many symbols for their internal representation.

An exception to the above rules on representation of rational functions occur when the denominator is purely numeric. It is then divided out to reduce the rational function to a single polynomial.

7.1.5 Representation of RF Names

RF names are internally represented by the P number or indirect name value converted to internal form followed by the name tag "T". Thus P5 is represented by ::+5:T where ::+5: is the internal representation of the literal polynomial 5. Similarly if VAR contains X, Y then P[J] is represented by ::+1J1:T while P[3X↑2+2] is represented by :X:+3X2:+2XO:T. Thus a SNOBOL statement of the form $'::+12:T'=:XY:+3X1Y2:-2XOY1:+10XOYO:
is the internal representation* of the ALGEBRA statement

P12=3XY↑2-2Y+10

7.2 Diagnostic Messages

This section gives a complete listing and explanation of all diagnostic messages generated in the ALGEBRA I system. The messages are listed alphabetically. Each message is followed by a listing of the source subroutine (in parenthesis), and a short comment about the error. For an explanation of source subroutine names and subroutine functions see Section 7.3.

When an error is detected the proper message(s) is printed on the teletype, if execution is in progress it is halted and the system returns the \ indicating it is awaiting further input.

---

*The indirection is required here because the symbol + is not allowed in explicit names in SNOBOL 67.

CANNØT DIFF. WITH RESPECT TØ A PAR. (DIF)

Differentiation allowed only with respect to letters currently on the global variable list VAR.

CANNØT INTEG. A RAT. FUN. (INTGR)

Integration is allowed for polynomials only.

CANNØT INTEG. WITH RESPECT TØ A PAR. (INTGR)

Integration is allowed only with respect to letters currently on the global variable list VAR.

CØNTENTS ØF SET ØF PARENTH. IS NULL! (CØNEI)

The form () has been found in the input polynomial.

DATA FØRMAT ERRØR (CØNEI)

Input polynomial or rational function contains an unbalanced parenthesis, on one of the sequences ++, +-, [,], --, -+, or ↑↑.

DECIMAL EXPØN. PRECLUDES SUBST. (SUB)

A number is being substituted into a variable or parameter which has a floating point exponent.

DIVISIØN NØT ALLØWED IN EXPØNENTS (CØNEI)

A division sign was found in an exponent.

DUPL. DEFN. (REXEC)

The label in the input line is the same as a previously used label.

DUPL. DEFN. RETYPE LABEL (REXEC)

The label used on an EDIT or LABEL is a duplication. Retype the label only on the next line.

EXPØNENTS AND NUMERIC CØEF. CANNØT HAVE EXPØNENTS (CØNEI)

An exponent expression or a numeric coefficient contains an exponent.

ILLEGAL CHAR. (REXEC)

One or more of the following characters appear in the input line in an illegal context.
?@←=&%$#"!:;⟨⟩

ILLEGAL EXIT (REXEC)

A procedure attempted to execute more lines than there were lines input because procedure runs off the end of saved statements without encountering and END statement.

ILLEGAL START (REXEC)

The START command can only be used to continue after a PAUSE inside of a procedure. To execute a block of code use the TØ command.

ILLEGAL TØ (REXEC)

The TØ command cannot be used inside a procedure. Use START.

ILLEGAL SUBST: VAR. LIST UNDEF. (REXEC)

A substitution cannot be made unless the variable list was explicitly given when the RF was named. A WITH modifier can still be used.

ILLEGAL STRUCTURE (REXEC)

An ALGEBRA statement has illegal syntax, an editing command has no blank spaces or too many blank spaces, or the line is completely unrecognizable as either a statement or a command.

INCØNSISTENT WITH (REXEC)

The WITH statement is attempting to replace the same letter or variable twice.

NEGATIVE ØR DECIMAL EXPØNENT PRECLUDES SUBST. ØF EXPR. (EXPA)

The coef. expression, variable or parameter into which a polynomial expression was substituted is raised to a negative or floating-point power and cannot be expanded.

PAR. USED AS VAR. (REXEC)

A letter previously used as a parameter is now being used as a variable.

PØWER ØF -1 PRECLUDES INTEGRATIØN (INTGR)

The variable of integration appears in the polynomial to the -1 power and cannot be integrated, since transcendental functions are not allowed in ALGEBRA.

SUBST. GENERATES DIV. IN EXPØN. (SUB)

The polynomial expression which is being substituted for a parameter in an exponent has a division in it.

SUBST. GENERATES EXPØN. IN EXPØN. (SUB)

The polynomial expression which is being substituted for a parameter in an exponent has parameter in it raised to a power greater than 1.

SUBST. GENERATES VAR. IN EXPØN. (SUB)

A polynomial expression containing a variable is being substituted for a parameter which appears in an exponent.

SUBST. ØF NULL VALUE ATTEMPTED (SUB)

A substitution of a null value for a variable or parameter has occurred. If a zero is intended it must be stated explicitly.

SUBST. PUTS CØEF. EXPR. INTØ EXPØN. (SUB)

The polynomial expression which is being substituted for a parameter in an exponent has a coefficient expression in it.

UNDEF. LABEL (REXEC)

An editing command or a goto uses an undefined label.

UNDEF. PØLY. USED (REXEC)

A polynomial name used in an expression has never been defined in previously executed statements.

UNDEF. VALUE (REXEC)

An error has occurred in the evaluation of an expression or polynomial. A message from the appropriate subroutine may precede this statement.

VARIABLES AND/ØR NESTED PARENTHESIS NØT ALLØWED IN EXPØNENTS (CØNEI)

A parenthesized exponent expression has nested parenthesis, or a variable is used in an exponent.

7.3 System Organization*

The ALGEBRA I system is made up of a set of packages written in SNOBOL 67. Most of the packages are written as SNOBOL 67 user defined functions* for ease in programming and to take advantage of the local garbage collection feature of SNOBOL 67 (see Ref. 1 for details). A complete list of the various packages showing names and calling arguments and a short description of their function is given below. Additional details on system organization, loading procedure, etc., well be published in another document. Some indication of the organization of the ALGEBRA I system and the calls between packages is indicated in Fig. 1. The unstarred functions in Fig. 1 are required in polynomial manipulations. The starred functions are additional requirements for manipulation of rational functions.

ADD(A,B)	add two RF; returns the sum.
ALPH(A,'B','C')	collect and delete the numeric terms in the alphanumeric string A: return this result in B, and the sum of the numeric terms in C.

*An understanding of SNOBOL 67 is assumed in this section.

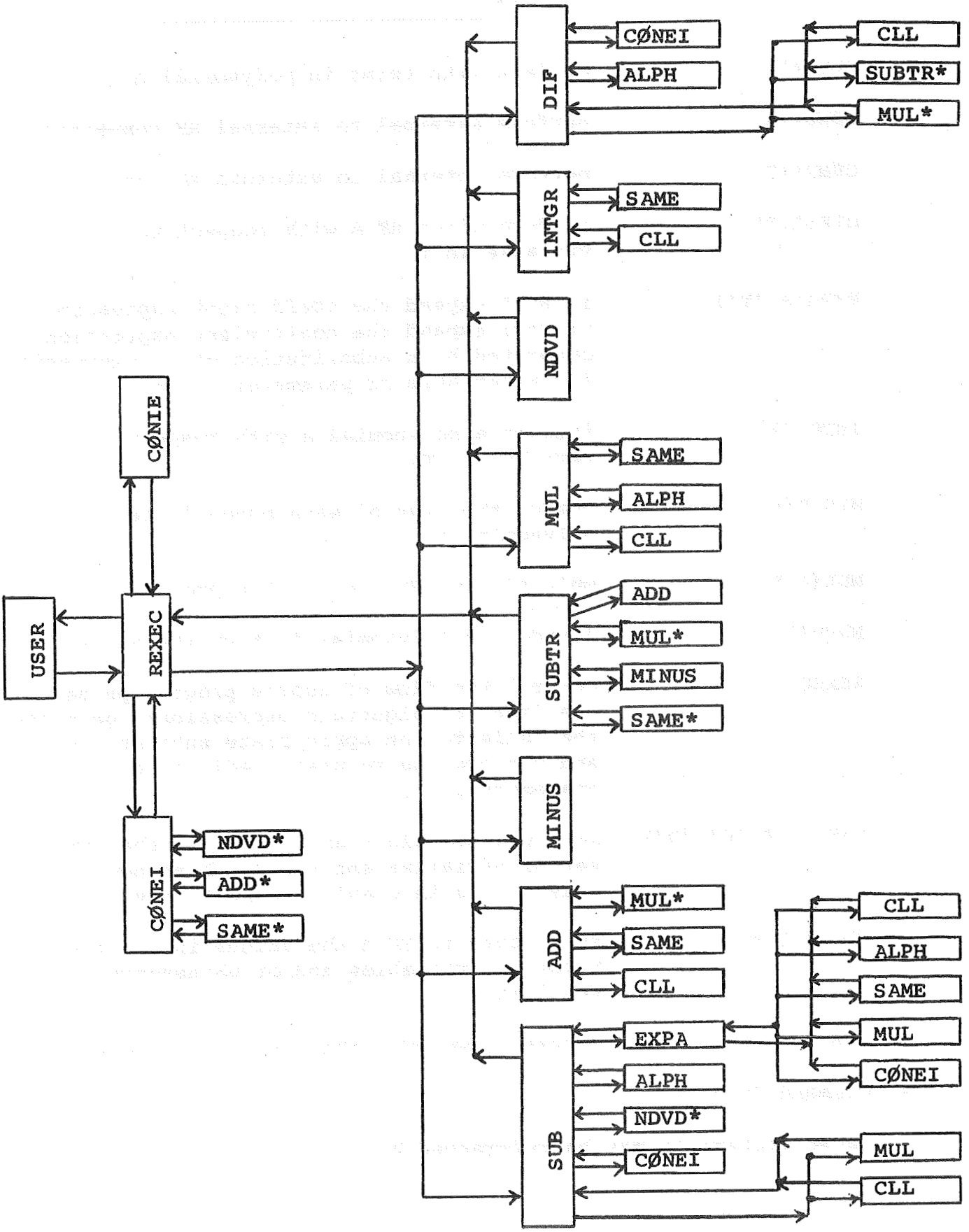


Fig. 1 ALGEBRA I System Organization

CLL(A)	collect like terms in polynomial A
CØNEI(A)	perform external to internal RF conversion
CØNIE(Z)	perform internal to external RF conversion
DIF(A,T)	differentiate RF A with respect to variable in T
EXPA(A,'T')	in RF A expand the coefficient expression if T=0; expand the coefficient expressions generated by a substitution of an expression for a variable or parameter if T=1.
INTGR(A,T)	integrate polynomial A with respect to variable in T.
MINUS(A)	change the sign of each monomial term in polynomial A.
MUL(A,B)	multiply two RF, return the product.
NDVD(A,N)	divide the polynomial in A by the number N.
REXEC	control the flow of user's program parse the lines of algebraic expressions, generate the calls to the appropriate subroutines and perform the requested editing of statements.
SAME(A,B,'C','D')	make polynomials A and B contain the same set of variables and return these new polynomials in C and D respectively.
SUB(A,T,W)	substitute in RF A the values in the list W for the variables and/or parameters in the list T.
SUBTR(A,B)	subtract two RF, return the difference.

7.4 Command Summary

Saved statements may be referenced by:

1. Statement number (statements numbered sequentially as saved). First statement saved is number 1.
2. By label or label + positive integer displacement.
3. By $ referring to the last line saved.

The commands available in ALGEBRA I are described below. In this description p and q are arbitrary statement references in any of the forms described above. Only the first letter of any command is required for recognition by the system. Additional letters are ignored.

A^{c}	delete an input character.
APPEND p	prints existing line and accepts addition by user, addition terminated with CR.
CONSTANTS	print present contents of global parameter list.
DELETE p	delete statement p.
DELETE p,q	delete statements p through q.
EDIT p	replace statement p with statement to be typed on next line.
LABEL p,⟨label⟩	add indicated label to beginning of line p or change label on line p to indicated label.
Q^{c}	delete an input line--restart in col. 1
READ	accept statements without immediate execution.
READ /file/	read statements from indicated file and append to existing saved statements.
START p	restart execution at p when p is inside a procedure.
TØ p	begin execution or restart (not in procedure) execution at p.

VARIABLES	print present contents of global variable list.
WRITE p,q ON /file/	write statements p through q on indicated disc file (word ON typed by the system).
X	execute statements immediately after input (normal mode).